

Lernzettel

Heaps und Prioritätswarteschlangen

Universität: Technische Universität Berlin
Kurs/Modul: Einführung in die Informatik - Vertiefung
Erstellungsdatum: September 20, 2025



Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

<https://study.AllWeCanLearn.com>

Einführung in die Informatik - Vertiefung

Lernzettel: Heaps und Prioritätswarteschlangen

(1) Grundidee. Ein Heap ist ein nahezu vollständiger Binärbaum, der eine Heap-Eigenschaft erfüllt:

- Max-Heap: Das Elternelement ist größer oder gleich seinen Kindern.
- Min-Heap: Das Elternelement ist kleiner oder gleich seinen Kindern.

Damit lässt sich die maximale (oder minimale) Priorität effizient abrufen.

(2) Varianten.

- Max-Heap: Priorität mit größter Zahl hat Vorrang.
- Min-Heap: Priorität mit kleinstem Wert hat Vorrang.

Diese beiden Varianten unterscheiden sich ausschließlich in der Richtung der Heap-Ebene.

(3) Repräsentation. Eine Heap-Array-Darstellung mit 0-basierter Indexierung verwendet folgende Beziehungen:

$$\text{left}(i) = 2i + 1, \quad \text{right}(i) = 2i + 2, \quad \text{parent}(i) = \left\lfloor \frac{i - 1}{2} \right\rfloor.$$

$A[0]$ ist das Wurzelement.

(4) Invariante.

- Max-Heap: Für alle $i > 0$ gilt
$$A[\text{parent}(i)] \geq A[i].$$
- Min-Heap: Für alle $i > 0$ gilt
$$A[\text{parent}(i)] \leq A[i].$$

Damit ist die Wurzel das aktuell größte bzw. kleinste Element.

(5) Grundoperationen.

- Einfügen (Insert):

Das neue Element wird am Ende eingefügt und dann nach oben verschoben (Sift-Up), solange es das Elternteil übertrifft (Max-Heap):

while $i > 0$ und $A[\text{parent}(i)] < A[i]$ gilt:

swap($A[i]$, $A[\text{parent}(i)]$) ; $i \leftarrow \text{parent}(i)$.

Komplexität: $O(\log n)$.

- Extrahiere-Min/Max (Delete-Max/Delete-Min):
 - Wurzelement mit dem letzten Element tauschen und das letzte entfernen.

- Von der Wurzel aus nach unten verschieben (Sift-Down/HeapifyDown), dabei das größere Kind (für Max-Heap) auswählen und tauschen, solange $A[i] < A[\text{Kind}]$.

wähle $j \in \{\text{left}(i), \text{right}(i)\}$ mit der größeren Kind-Wertung.

if $A[i] < A[j]$ swap ($A[i], A[j]$); fortfahren .

Komplexität: $O(\log n)$.

(6) Build-Heap. Aus einem beliebigen Array A lässt sich in $O(n)$ ein Heap erzeugen, indem man von den letzten Nicht-Blattknoten bis zum Wurzelknoten heapifiziert:

BuildHeap(A) : for $i = \lfloor \frac{n}{2} \rfloor - 1$ downto 0 : HeapifyDown(A, i).

(7) Prioritätswarteschlange. Eine Priority Queue, implementiert via Heap, unterstützt typischerweise:

insert(p, w), extract_top(), peek_top(), change_priority($\cdot$).

Dabei entspricht der Wert w der Priorität des Elements p . Die effiziente Umsetzung erfolgt in Java oder einer anderen Sprache über ein Heap-basiertes Interface.

(8) Beispiel (intuitive Skizze). Gegeben ein Max-Heap-Hinweis-Array: $[9, 7, 6, 5, 3, 4]$ ergibt sich nach einem Insert von 8: $[9, 8, 6, 5, 3, 4, 7]$ und nach einem Extract-Max wieder $[8, 7, 6, 5, 3, 4]$.

Hinweise zur Implementierung in Java (kurz).

- Nutze ein Array oder eine ArrayList als zugrunde liegende Struktur.
- Implementiere left, right, parent als Hilfsfunktionen.
- Achte darauf, dass der Heap immer eine nahezu vollständige Baumstruktur bleibt.